An Improved Learning Algorithm for Augmented Naive Bayes

Huajie Zhang and Charles X. Ling

Department of Computer Science
The University of Western Ontario
London, Ontario, Canada N6A 5B7
{hzhang, ling}@csd.uwo.ca

Abstract. Data mining applications require learning algorithms to have high predictive accuracy, scale up to large datasets, and produce comprehensible outcomes. Naive Bayes classifier has received extensive attention due to its efficiency, reasonable predictive accuracy, and simplicity. However, the assumption of attribute dependency given class of Naive Bayes is often violated, producing incorrect probability that can affect the success of data mining applications. We extend Naive Bayes classifier to allow certain dependency relations among attributes. Comparing to previous extensions of Naive Bayes, our algorithm is more efficient (more so in problems with a large number of attributes), and produces simpler dependency relation for better comprehensibility, while maintaining very similar predictive accuracy.

1 Introduction

Learning Bayesian classifier is a process of constructing a special Bayesian network from a given set of pre-classified examples, each of which is represented by a vector of attribute values. Assume A_1, A_2,..., A_n are n attributes which take values $a_1, a_2, , ..., a_n$ respectively. Those attributes will be used collectively to predict the value c of another attribute C, called class label.

According to the Bayesian rule, the probability of an example E being in class c is:

$$p(C = c|a_1, a_2, ..., a_n) = \frac{p(a_1, a_2, ..., a_n|C = c)p(C = c)}{p(a_1, a_2, ..., a_n)}$$

The classification is taken as the C's value with the largest probability.

Assume all attributes are independent given the class. That is:

$$p(a_1, a_2, ..., a_n|c) = p(a_1|c)p(a_2|c)...p(a_n|c)$$

The resulting Bayesian classifier is called the *Naive Bayesian classifier*.

Where strong dependent relations do exist among attributes (such as $A_i = A_j$), the probability of Naive Bayes will not be correct. Friedman (1997) presented his work on learning tree-like Bayesian networks, TAN (Tree-Augmented

Naive Bayes), in which non-classification attributes can form a tree structure. A TAN is a compromised representation between a full Bayesian network and Naive Bayes.

The basic idea is to approximate the underlying probability distribution by conditional mutual information. Its time complexity is $O(n^2)$, where n is the number of the attributes. [1] Keogh and Pazzani (1999) present another approach to learn TANs, which searches heuristically for a TAN guided by the predictive accuracy. Their algorithm, called *SuperParent*, also has the time complexity $O(n^2)$. They show that their algorithm consistently predicts more accurately than Friedman (1997)'s TAN, which in turn, predicts more accurately than the Naive Bayes. *SuperParent* consists of two major steps. The first step searches for a best super parent that improves the predictive accuracy the most. A super parent is a node with arcs pointing to all other nodes without a parent (not counting the class label). The second step determines one best child for the super parent chosen at the first step, again based on the predictive accuracy. After this iteration of the two steps, one arc is added on the TAN, and this process repeats until no improvement is achieved, or $n-1$ arcs are added into the tree. Obviously, *SuperParent* is a greedy algorithm with complexity $O(n^2)$. Our algorithm, called *StumpNetwork*, is constant-factor faster than *SuperParent*, while maintaining very similar predictive accuracy. The constant becomes larger in domains with a large number of attributes. This speed-up is important for data mining problems with a large number of attributes.

2 An Improved Algorithm for Learning TANs

We extend Keogh and Pazzani (1999) by proposing a more efficient algorithm, called *StumpNetwork*, to construct a special class of TAN. The motivation of *StumpNetwork* derives from the observation that the dependence among attributes tends to cluster into groups in many read-world domains with a large number of attributes. Attributes in each cluster form a simple dependency relation: a one-level tree structure, called the tree stump.[2] For example, in the customer database of a commercial bank, the amount of deposit, credit, and debt may be dependent on customer's total income, while the house price, utility bills, and neighbourhood are dependent on the postal code. But those tree stumps may not be totally independent, so some simple dependency relations are allowed among tree stumps. That it, we search for a special class of the tree structure as the topology of our Bayesian network.

Our algorithm constructs simple tree stumps first, and then construct links among the tree stumps. Similar to SuperParent (Keogh and Pazzani, 1999), search in *StumpNetwork* is guided by the predictive accuracy. That is, our cri-

[1] The time complexity is also linear to the size of the training set. But this term is same for all Augmented Naive Bayesian classifiers in our discussion, so we omit it in our paper.

[2] Holte (1993) studied tree stumps as decision tree classifiers.

terion of constructing tree stumps and links between tree stumps is solely to improve the predictive accuracy of the Augmented Naive Bayesian classifier.

2.1 *StumpNetwork*

We first define the kind of tree structures as our hypothesis space, and then describe greedy search strategies for finding such tree structures.
Definition 1: A tree $T(r, N)$, where r is an attribute and N is a set of attributes, is called a *tree stump*, if the tree is of root r and of height 1, and there is an arc from r to each node in N.
Definition 2: A set of tree stumps is called Stump Network if the intersection of any two tree stumps is empty, and tree stumps may be connected in such a way that the root of one tree stump is pointed by at most a leaf node of another tree stump.

Figure 1 shows an example of a Stump Network. Clearly Stump Network is a special form of the tree topology where attribute dependencies tend to form clusters.

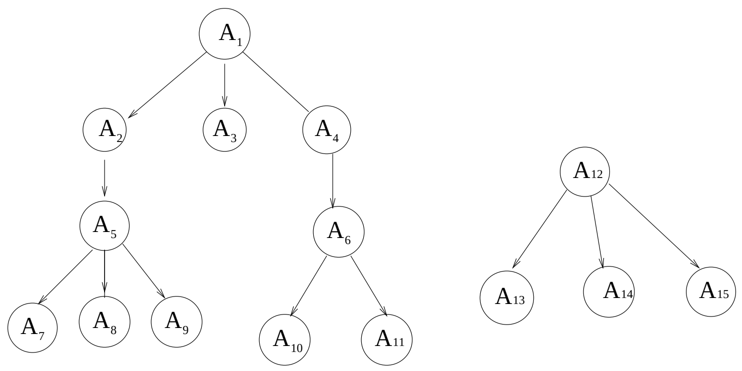

Fig. 1. An example of a Stump Network, a special class of TAN. Note that classification node C and all links from C to all A_i for all i are omitted for simplicity.

Our algorithm for searching Stump Network heuristically consists of two main steps. In the first step, it finds out a set of tree stumps, if adding such tree stumps into the Naive Bayes improves the the predictive accuracy on a testing set. The resulting set of tree stumps are sorted by the improvement of the predictive accuracy. The second step goes through the sorted list of the tree stumps once, to see if each tree stump can be pointed by a leaf of the previous tree stumps in the sorted list that will result in an improved predictive accuracy. If so, such a link will be remained in the Stump Network. The algorithm called *StumpNetwork* is presented below.

1. Read the data D

2. Initialize B to be Naive Bayes and calculate its predictive accuracy
3. Let node set N be all attributes (except C)and *TreeStump* queue be empty
4. Make a tree stump for each node in N. Let T_s be the tree stump with the highest improvement on the predictive accuracy
5. For each arc on T_s, if the predictive accuracy does not decrease after deleting the arc, remove it from T_s
6. Put T_s in *TreeStump* queue
7. Remove all nodes of T_s from N. If N is not empty, go to 4
8. Go over *TreeStump* queue once, for each tree stump in the queue, add a link from a leaf of the previous tree stumps in the queue to the root of this tree stump, if the predictive accuracy increases

Similar to *SuperParent*, our algorithm finds one super parent that links to all other possible attributes with the highest improvement on the predictive accuracy (Step 4). However, unlike *SuperParent* which keep just one child of the parent node found in the previous step (thus *SuperParent* needs to loop n times), we keep all the child nodes they do not result in a decreased predictive accuracy (Step 5). The resulting one-level tree forms a tree stump. Tree stumps may be linked together in Step 8.

2.2 Theoretical comparison of *StumpNetwork* to *SuperParent*

From the description of the *StumpNetwork* algorithm above, the time for the creation of the tree stumps (from *step 4* to *step 7*) is $O(n^2)$ and the time for *step 8* is $O(n)$, where n is the number of attributes. Therefore, we consider the time complexity of constructing tree stumps as the time complexity of *StumpNetwork* T_c. We have:

$$T_c(n) = 2n + 2(n - m_1) + 2(n - m_1 - m_2) + ... + 2(n - \sum_{i=1}^{N-1} m_i) \tag{1}$$

where N is the number of the tree stumps, and m_i is the size of the tree stump i. Let $k = \sum_{i=1}^{N} m_i$. We have:

$$T_c(n) = \frac{n(n+k)}{k} \tag{2}$$

Because the tree stump found in Step 4 improves the predictive accuracy, and deleting edges in Step 5 will not decrease the predictive accuracy, the resulting tree cluster must have its size greater than 1. That is, $k \geq 2$.

It is easy to obtain the time complexity of *SuperParent*, T_s, as (Keogh and Pazzani 1999):

$$T_s(n) = 2n + 2(n-1) + ... + 1 = n(n+1) \tag{3}$$

We omit the items of $O(n)$ in Equation 2 and Equation 3, then we have:

$$\frac{T_c(n)}{T_s(n)} = \frac{1}{k} \tag{4}$$

Since $k \geq 2$, $T_c(n) \leq 1/2T_s(n)$. The greater the k, the less $T_c(n)$ than $T_s(n)$.

In many data mining applications, the number of attributes is often quite large (hundreds to thousands). In addition, those attributes can be clustered into several large groups. For these problems, our algorithm will be many times more efficient than *SuperParent*. The greater of the number of the attributes in each group, the more efficient our algorithm will be.

The empirical results shown in the next section verify this conclusion. However, little is lost with a simplified TAN and an improved efficiency: the predictive accuracies of the two algorithms will be shown to be very similar in the next subsection.

2.3 Experimental Results of *StumpNetwork*

We compare *StumpNetwork* with *SuperParent* using several datasets from the UCI repository (Merz, 1997), and one real-world dataset from a bank that we worked on in a previous project. Table 1 lists the properties of the datasets we used in our experiment.

Dataset	Attributes	Class	Instances
Ecoli	7	8	336
Vote	16	2	435
Pima	8	2	768
Australia	14	2	690
Breast	10	2	683
Segment	19	7	1540
Vehicle	18	4	846
Bank	20	2	1162

Table 1. Descriptions of domains used in our experiments.

Our experiments follow the procedure below:

1. The continuous attributes in the dataset are discretized by Fayyad(1993)'s entropy-based method.
2. Calculate the average predictive accuracy of *SuperParent* and *StumpNetwork* respectively, with 5 fold cross validation.

Table 2 shows the experimental results. As we can see, *StumpNetwork* achieves essentially the same testing accuracy as *SuperParent*. On the other hand, the time used in constructing Augmented Naive Bayes is quite different. On average over the datasets we compared, *StumpNetwork* is 4.8 times faster than *SuperParent*. There is a clear trend of larger saving with larger datasets and datasets with a larger number of attributes.

Dataset	*StumpNetwork*		*SuperParent*	
	Accuracy	**Time**	**Accuracy**	**Time**
Ecoli	84.43±1.07	4.08	83.11±1.16	7.74
Vote	96.07±1.99	16.33	94.02±2.23	39.63
Pima	78.98±2.28	2.75	79.39±1.83	6.36
Australia	85.43±1.83	17.97	86.07±1.47	53.59
Breast	96.38±0.70	37.17	96.56±0.91	97.61
Segment	93.37±0.96	1675.80	94.55±0.78	8275.60
Vehicle	69.10±1.86	152.39	68.91±1.13	668.55
Bank	53.22± 1.62	348.37	53.58±1.94	1726.20
Average	81.87	281.9	82.02	1359.4

Table 2: Comparison of our *StumpNetwork* and *SuperParent*.

3 Conclusions

In this paper we present a new algorithm for constructing a special kind of tree augmented Naive Bayes based on work by Keogh and Pazzani (1999). Our algorithm works best in domains with a large number of attributes, and attributes tend to form large clusters with simple dependency relations inside clusters. Both experimental and theoretical analyses show that our algorithm is constant-factor faster than Keogh and Pazzani (1999)'s algorithm, and the constant is proportional to the size of the clusters. Our algorithm also produces simpler tree structure than an arbitrary tree, thus producing more comprehensible results. Empirical comparisons demonstrate that both algorithms have very similar predictive accuracies.

In our future research, we will study other efficient and specialized Augmented Naive Bayes suitable for domains possessing certain properties commonly occurring in real-world applications.

Acknowledgements

This research is based on the work of Mr. Keogh and Pazzani and Dr. Pazzani. They kindly provided us with the source codes of *SuperParent*, which is a great help to us.

References

[1997] Friedman, N. : Bayesian network classifier. Machine Learning, **29** (1997) 131–161.

[1993] Holte, R. : Very simple classification rules perform well on most commonly used datasets. Machine Learning, **3** (1993), 63–91.

[1999] Keogh and Pazzani , E. , Pazzani, M. : Learning augmented bayesian classifiers. Proceedings of Seventh International Workshop on AI and Statistics. (1999) Ft. Lauderdale.

[1997] Merz, C.: Uci repository of machine learning databases. Dept. of Information and Computer Science, University of California, (1997) Irvine.